Manuscript received Jan. 23, 2007; revised April 18, 2007

Specification and Analysis of Hardware Systems Using Timed Process Algebras

K.L. Man
Centre for Efficiency-Oriented Languages (CEOL)
Department of Computer Science
University College Cork (UCC)
Ireland
pafesd@gmail.com

Abstract: The ability of unambiguously specifying (in a mathematical sense) and rigorously analysing timing properties/constraints is fundamental to design correct hardware systems. Formalisms in which hardware behaviour and timing properties can be precisely captured is a mandatory prerequisite for designing correct hardware systems (discrete-time systems by nature). Timed process algebras are such formalisms. To show that timed process algebras are useful for formal specification and analysis of hardware systems, in this paper, we illustrate the use of a timed process algebra called timed Chi (χ) with several benchmark examples of hardware systems.

Key–Words: Formal languages, Formal semantics, Process algebras, Real-time systems, Formal specification and analysis, Hardware systems

1 Introduction

Formal languages with a semantics formally defined in *Computer Science* increase understanding of systems, increase clarity of specifications and help solving problems and remove errors. Over the years, several flavours of formal languages have been gaining industrial acceptance.

Process algebras [1] are formal languages that have formal syntax and semantics for specifying and reasoning about different systems. They are also useful tools for verification of various systems. Generally speaking, process algebras describe the behaviour of processes and provide operations that allow to compose systems in order to obtain more complex systems. Moreover, the analysis and verification of systems described using process algebras can be partially or completely carried out by mathematical proofs using equational theory.

In addition, the strength of the field of process algebras lies in the ability to use *Algebraic Reasoning* (also known as equational reasoning) that allows rewriting processes using axioms (e.g. for commutativity and associativity) to a simpler form. By using axioms, we can also perform calculations with processes. These can be advantageous for many forms of analysis.

Serious efforts have been made in the past to deal with systems (e.g. real-time systems [2, 3] and hybrid systems [4, 6]) in a process algebraic way. Over the years, process algebras have been successfully used in a wide range of problems and in practical applications in both academia and industry for analysis of many different systems.

On the other hand, the need for a formal and well-defined semantics of a hardware description language is widely accepted and desirable for architects, engineers and researchers in the electronic design community.

The ability of unambiguously specifying (in a mathematical sense) and rigorously analysing timing properties/constraints is fundamental to design correct hardware systems. Formalisms in which hardware behaviour and timing properties can be precisely captured is a mandatory prerequisite for designing correct hardware systems. Timed process algebras are such formalisms.

For illustration purpose, in this paper, we choose the timed process algebra called timed Chi (χ) [5] as the main reference timed process algebra for the use of specification and analysis of hardware systems. This particular choice is immaterial and other timed process algebras may be used as well.

The timed Chi formalism is obtained by means of the simplification of hybrid Chi formalism [6, 7]. Principally, the timed Chi formalism is suited to modelling, simulation, verification and real-time control. Its application domain consists of large and complex manufacturing systems.

The formal semantics of timed Chi is defined by

means of deduction rules in a *Structured Operational Semantics* (SOS) style [8] that associate a time transition system with a timed Chi process. A set of axioms/properties of timed Chi is presented for a notion of equivalence (bisimulation). The straightforward syntax and semantics is also highly suited to architects, engineers and researchers from the hardware design community.

To show that timed Chi is also well-suited for addressing various aspects of hardware systems, in this paper, we illustrate the use of timed Chi with some benchmark examples of hardware systems: a D flip-flop, an asynchronous arbiter and a simple arbiter (with assertion).

In the remainder of this paper, we may usually refer to timed Chi as χ. Also, we may write formal χ specification as χ specification.

1.1 Related work

In the hardware design community, architects use hardware description languages (that are not defined by means of mathematics) like Verilog [9] and VHDL [10] to model hardware systems. Since these description languages are not formally (i.e. mathematically) defined, the models described using them may be ambiguous.

It is worth mentioning that, after these description languages have been widely used, some research works on their formal semantics, except some trails (e.g. [11]), are mostly based on *Abstract State Machine* (ASM) specifications and *Denotational Semantics* (e.g. [12, 13, 14]) that have been done by other researchers.

Hence, we believe that there is a gap between the intuition behind the description languages (given by the developers) and the formal semantics of the description languages defined by the researchers. Currently, the analysis of hardware systems (e.g. modelled in Verilog and VHDL) is mainly addressed by a simulation context.

Simulation engineers apply the simulators, that are built based on their semantics (that is not formally defined), to simulate the behaviour of such systems. The results are then not always guaranteed to be correct.

Furthermore, it is generally believed that the SOS style semantics is more intuitive, and the methods of ASM specifications and denotational semantics appear to be difficult to apply to describe the dynamic behaviour of processes.

Since processes are the basic units of execution for simulating the behaviour of a device or a system within languages (for hardware systems), process algebras with the SOS style semantics are potentially a good candidate for giving formal specifications of systems in the hardware design community.

Note that some comparisons and related work of formalisms with timing can already be found in [19] and [20].

1.2 Paper organisation

This paper is organised as follows. In Section 2, we give a brief overview of the χ formalism. Through some simple examples, Section 3 shows that the deduction rules of χ can ensure the correctness of specifications and can help modellers to make correct specifications. Some samples (modelling several benchmark hardware systems) of the applications of χ are shown in Section 4.

A variety of approaches that can be used for the analysis of the formal specifications described in χ is presented in Section 5. Finally, concluding remarks are made in Section 6 and the direction of future work is pointed out in the same section.

2 χ Formalism

χ is such a rich formalism and presenting the complete formal syntax and formal semantics of it is far beyond the scope of this paper. Hence, in this section, we informally present just a small part of χ, disregarding features[1] that may not be relevant for the use in this paper.

Again, in what follows, we refer to this small part of timed χ as χ. For an extensive treatment of χ, the reader is referred to [5].

2.1 Data types, time model, synchronisation and communication model

2.1.1 Data types

χ is statically strongly typed. Every variable has a type which defines the allowed values of that variable and the allowed operations on that variable. The basic types are natural numbers, integers, real numbers, booleans, strings and enumerations.

Type constructors operate on existing types to create structured types. χ uses type constructors to create sets, lists, array tuples, record tuples, dictionaries, functions and distributions (for stochastic models).

Channels also have a type that indicates the type of data that is communicated via the channel. Pure

[1] For instance, recursive definitions, operators used for scoping and communication, process definition and process instantiation are not treated in this paper.

synchronisation channels, that do not communicate data, are of the predefined type void.

2.1.2 Time model

The time in χ is dense. So, timing is measured on a continuous time scale. χ has a strong time determinism principle. This means that passage of time cannot result in making a choice between the two operands of the choice.

Also, the maximal progress (a process can delay only if it cannot do anything else) is not implicit in χ.

In any χ specification, the existence of the predefined reserved global variable `time` which denotes the current time, the value of which is initially zero, is assumed.

2.1.3 Synchronisation and communication model

In χ, the synchronisation and communication mechanism are based on CSP [21, 18]. This means that, although a channel can be used in any number of processes, synchronisation or communication always occurs on a point-to-point basis, i.e. synchronisation or communication always occurs between exactly two processes.

Also, this synchronisation or communication mechanism is widely used in the hardware design community for modelling asynchronous circuits.

2.2 χ specification

A χ specification (restricted to the use in this paper) is of the following form:

$$
\begin{array}{l}
\langle\ \mathsf{disc}\ s_1, \ldots, s_k \\
,\ \mathsf{chan}\ h_1, \ldots, h_l \\
,\ i \\
\mid\ p \\
\rangle
\end{array}
$$

where

- $s_1, \ldots, s_k$ denote the discrete variables;
- $h_1, \ldots, h_l$ denote the channels;
- i denotes an initialisation predicate that restricts the allowed values of the variables initially;
- p is a process term defining the behaviour of the specification.

Notice that the keywords disc and chan are omitted where there are no discrete variable declarations and are no channel declarations, respectively. Also, the initialisation predicate may be omitted, indicating a predicate that always holds.

The set P of process terms $p \in P$ (for the use in this paper) is defined according to the following grammar:

$$
\begin{array}{rcl}
p & ::= & \mathbf{x}_n := \mathbf{e}_n \mid \Delta d \mid [p] \mid p;\ p \\
& \mid & b \rightarrow p \mid p \,[\!]\, p \mid p \parallel p \mid h\,!!\,\mathbf{e}_n \\
& \mid & h\,??\,\mathbf{x}_n \mid \partial_A(p) \mid *p
\end{array}
$$

Here, $\mathbf{x}_n$ and $\mathbf{e}_n$ are a list of variables $x_1, \ldots, x_n$ and a list of expressions $e_1, \ldots, e_n$, respectively. $d \in \mathbb{R}_{\geq 0}$ and b denotes a guard (i.e. a boolean expression). Moreover, h is a channel and A represents a set of actions.

In χ, it is allowed to use common arithmetic operators (e.g. $+, -$), relational operators (e.g. $=, \geq$) and logical operators (e.g. $\wedge, \vee$) as in mathematics to construct expressions over variables.

The operators are listed in descending order of their binding strength as follows $\rightarrow$, ; , $\{\parallel, [\!]\}$. The operators inside the braces have equal binding strength. In addition, operators of equal binding strength associate to the right, and parentheses may be used to group expressions. For example, $p;\ q;\ r$ means $p;\ (q;\ r)$, where $p, q, r \in P$.

2.3 Concise explanation of the syntax

2.3.1 Atomic process terms

The atomic process terms of χ are undelayable process term constructors that cannot be split into smaller process terms. They are:

1. The *multi-assignment* process term $\mathbf{x}_n := \mathbf{e}_n$. It assigns the values of expressions $e_1, \ldots, e_n$ to variables $x_1, \ldots, x_n$, respectively, in an atomic way.
2. The *send* process term $h\,!!\,\mathbf{e}_n$. It sends the values (must be defined) of expressions $e_1, \ldots, e_n$ via channel h by means of internal send actions.
3. The *receive* process term $h\,??\,\mathbf{x}_n$. It receives values (of size n) via the channel h and assigns them to the variables $x_1, \ldots, x_n$ by means of internal receive actions.

2.3.2 Operators

Atomic process terms can be combined using the following operators. The operators are:

1. The *delay operator* Δd denotes a process term that first delays for d time units, and then terminates by means of the internal action τ.

2. The *sequential composition* of process terms p and q (i.e. $p;\ q$) behaves as process term p until p terminates, and then continues to behave as process term $q \in P$.

3. By means of the *any delay operator* $[p]$, delay behaviour of arbitrary duration can be specified. The resulting behaviour is such that arbitrary delays are allowed. As a consequence, any delay behaviour of p is neglected. The action behaviour of p remains unchanged.

4. The *guarded process term* $b \rightarrow p$ can perform whatever actions p can perform under the condition that the guard b (a boolean expression) evaluates to true in the current state. The guarded process term can delay according to p under the condition that the guard b holds. The guarded process term can perform arbitrary delays under the condition that the guard b does not hold.

5. The *alternative composition* of process terms p and q (i.e. $p \ [\!] \ q$) allows a non-deterministic choice between different actions of the process term either p or q. With respect to time behaviour, the participants in the alternative composition have to synchronise.

6. The *parallel composition* of process terms p and q (i.e. $p \parallel q$) executes p and q concurrently in an interleaved fashion with the possibility of synchronisation or communication (in a CSP based) between p and q. Also, with respect to time behaviour, the participants in the parallel composition have to synchronise.

7. The *encapsulation operator* $\partial_A(p)$ is introduced to block the actions that p can perform from the set A.

8. The *repetition* process term $*p$ represents the infinite repetition of process term p.

2.4 Formal semantics of χ

This subsection informally describes the formal semantics of χ. It is defined by means of deduction rules in SOS style that associate a time transition system with a χ process. Three different kinds of transition relations are defined namely:

1. one associated with termination transition;

2. one associated with action transition (for discrete action);

3. one associated with time transition (delay behaviour).

3 Correctness of χ Specifications

As we already mentioned in Section 2 that the formal semantics of χ is defined by means of deduction rules in SOS style. These deduction rules ensure the correctness of χ specifications and can help modellers to make correct specifications.

In this section, for illustration purposes, we define some deduction rules (only for the use in this section to give the first impression of such deduction rules to the reader) and show their use through some toy examples in χ.

For those who have a Computer Science background, this section can be left out.

3.1 Deduction rules

Here, we define several deduction rules for atomic process term: multi-assignment; and operators: sequential composition, alternative composition and parallel composition.

For the set P of process terms $p \in P$ (for the use in this subsection), we have:

$$p ::= \ \mathbf{x}_n := \mathbf{e}_n \ \mid \ p;\, p \ \mid \ p \ [\!] \ p \ \mid \ p \parallel p$$

We further define the following deduction rules:

$$\frac{}{\mathbf{x}_n := \mathbf{e}_n \rightarrow \checkmark}\ 1 \qquad \frac{p \rightarrow \checkmark}{p;\, q \rightarrow q}\ 2 \qquad \frac{p \rightarrow \checkmark}{p \ [\!] \ q \rightarrow \checkmark}\ 3$$

$$\frac{p \rightarrow \checkmark}{q \ [\!] \ p \rightarrow \checkmark}\ 4 \qquad \frac{p \rightarrow \checkmark}{p \parallel q \rightarrow q}\ 5 \qquad \frac{p \rightarrow \checkmark}{q \parallel p \rightarrow q}\ 6$$

The above deduction rules (of the form $\frac{premise}{conclusion}$) have two parts: on the top of the bar we put premise of the rule, and below it the conclusion. If the premise holds, then we infer that the conclusion holds as well. Moreover, $\rightarrow$ and $\checkmark$ are used to represent a transition and a terminated process respectively.

Rule 1 states that $\mathbf{x}_n := \mathbf{e}_n$ can always perform a transition to a terminated process (i.e. successful termination).

The sequential composition of the process terms p and q (i.e. $p;\ q$) behaves as process term p until p terminates, and then continues to behave as process term q (see Rule 2).

The effect of applying the alternative operator to the process terms p and q (i.e. $p \ [\!] \ q$) is that the execution of a transition by either one of them results in a definite choice as shown in Rules 3 and 4.

The parallel composition of the process terms p and q (i.e. $p \parallel q$) has as its behaviour with respect to transitions the interleaving of the behaviours of p and q (see Rules 5 and 6).

3.2 Running examples

Using the above deduction rules, for instance, we can prove that:

1. process term $\mathbf{x}_n := \mathbf{e}_n;\ (\mathbf{x'}_n := \mathbf{e'}_n \ [\!] \ \mathbf{x''}_n := \mathbf{e''}_n)$ can terminate successfully after a finite number of transitions.

   - ***Proof:*** According to Rule 1, $\mathbf{x}_n := \mathbf{e}_n$ can always perform a transition to a terminated process. Due to this, we can apply Rule 2 to obtain $\mathbf{x}_n := \mathbf{e}_n;\ (\mathbf{x'}_n := \mathbf{e'}_n \ [\!] \ \mathbf{x''}_n := \mathbf{e''}_n) \rightarrow \mathbf{x'}_n := \mathbf{e'}_n \ [\!] \ \mathbf{x''}_n := \mathbf{e''}_n$. Using Rule 1 together with either Rule 3 or Rule 4, we can further have $\mathbf{x'}_n := \mathbf{e'}_n \ [\!] \ \mathbf{x''}_n := \mathbf{e''}_n \rightarrow \checkmark$.

2. process term $(\mathbf{x}_n := \mathbf{e}_n \parallel \mathbf{x'}_n := \mathbf{e'}_n);\ \mathbf{x''}_n := \mathbf{e''}_n$ cannot terminate successfully in two transitions.

   - ***Proof:*** We assume to have $(\mathbf{x}_n := \mathbf{e}_n \parallel \mathbf{x'}_n := \mathbf{e'}_n);\ \mathbf{x''}_n := \mathbf{e''}_n \rightarrow \mathbf{x''}_n := \mathbf{e''}_n \rightarrow \checkmark$ according to Rules 1 and 2. This means that we must have the transition $\mathbf{x}_n := \mathbf{e}_n \parallel \mathbf{x'}_n := \mathbf{e'}_n \rightarrow \checkmark$ as a premise necessarily. However, this is not possible due to Rules 5 and 6.

3.3 Properties

We can also deduce some properties (that add to the level of confidence one has with respect to the correctness of the formal semantics) for all specifications that can be generated by the set P of process terms according to the deduction rules as defined in Subsection 3.1. For instance, we can have the following properties for equivalence:

- $p \ [\!] \ q = q \ [\!] \ p$ and $p \parallel q = q \parallel p$ (so-called commutativity property);
- $p@(q@r) = (p@q)@r$, for $@ \in \{;\ ,\ [\!],\ \parallel\}$ (so-called associativity property).

Using properties, we can rewrite a χ specification to a simpler form. This can be advantageous for many forms of analysis. As we already mentioned, in the field of process algebras, this is so-called algebraic reasoning.

4 Hardware Systems in χ

This section presents some samples (modelling several benchmark hardware systems) of the applications of χ.

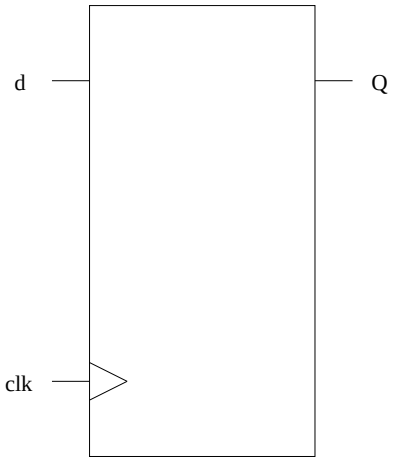

Figure 1: A D flip-flop.

4.1 A D flip-flop

D flip-flops are among the basic building blocks of *Register-Transfer Level* (RTL) designs. A D flip-flop has a clock input (clk) in the sensitivity list, a data input (d) and a data output (Q).

When a positive or negative edge occurs in the clock signal (which means that $(\mathrm{clk} \wedge \neg\mathrm{clk}^-) \vee (\mathrm{clk}^- \wedge \neg\mathrm{clk})$), the value of input port d is assigned to output port Q. Note that x^- denotes the value of variable x before execution of an discrete action.

Figure 1 depicts such a D flip-flop. A χ specification is given as follows:

$$
\begin{array}{l}
\langle\ \mathsf{disc}\ \mathrm{d}, \mathrm{Q}, \mathrm{clk} \\
,\ \mathrm{d} = \mathrm{true}, \mathrm{Q} = \mathrm{true}, \mathrm{clk} = \mathrm{false} \\
|\ *(\mathrm{DFF} \parallel (\mathrm{CLK_a} \ [\!] \ \mathrm{CLK_7})) \\
\rangle,
\end{array}
$$

$$
\begin{array}{c}
\mathrm{DFF} \approx (\mathrm{clk} \wedge \neg\mathrm{clk}^-) \vee (\mathrm{clk}^- \wedge \neg\mathrm{clk}) \rightarrow \mathrm{Q} := \mathrm{d}, \\
\mathrm{CLK_a} \approx [\mathrm{SWITCH};\ \mathrm{INPUT}], \\
\mathrm{CLK_7} \approx \Delta 7;\ \mathrm{SWITCH};\ \mathrm{INPUT}, \\
\mathrm{SWITCH} \approx \neg\mathrm{clk}^- \rightarrow \mathrm{clk} := \mathrm{true} \ [\!] \ \mathrm{clk}^- \rightarrow \mathrm{clk} := \mathrm{false}, \\
\mathrm{INPUT} \approx \mathrm{d} := \mathrm{true} \ [\!] \ \mathrm{d} := \mathrm{false}.
\end{array}
$$

In the χ specification, clk, d and Q are modelled by boolean variables. The complete system is modelled by a repetition of the parallel composition of process terms DFF and a choice between process terms $\mathrm{CLK_a}$ and $\mathrm{CLK_7}$.

The process term DFF describes the behaviour of the D flip-flop. When a positive or negative edge occurs in clk, the value of d is assigned to Q. Otherwise, it performs an arbitrary delay.

The process terms $\mathrm{CLK_a}$ and $\mathrm{CLK_7}$ model the behaviour of the clock clk with the frequency of arbitrary and 7 time units respectively. The switching

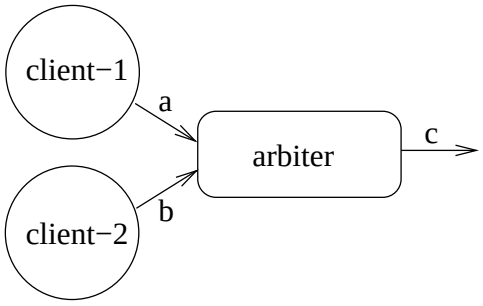

Figure 2: An asynchronous arbiter.

from positive edge to negative edge or vice versa is modelled by the process term SWITCH.

A non-deterministic choice of the clock frequency (any value) for at most 7 time units is assigned to clk by means of the alternative composition of $\mathrm{CLK_a}$ and $\mathrm{CLK_7}$ (i.e. $\mathrm{CLK_a} \,[\!]\, \mathrm{CLK_7}$).

In the process term $\mathrm{CLK_a}$, the any delay operator $[\,]$ is needed to apply to SWITCH; INPUT, because otherwise $\mathrm{CLK_a} \,[\!]\, \mathrm{CLK_7}$ may not delay together for at most 7 time units (as already explained in Section 2 that χ a has strong time determinism principle).

For simulation purposes, a test-bench of d (assigning a value "true" or "false" to d arbitrarily) is given by the process term INPUT.

4.2 An asynchronous arbiter

Asynchronous arbiter circuits are standard hardware verification benchmark circuits. An arbiter circuit controls the exclusive access of one out of a number possibly competing processes to a shared resource.

Figure 2 shows an (untimed) asynchronous arbiter (taken from [15]) such that two clients (client-1 and client-2) complete for a shared resource. Each client sends a request (a number 1 for client-1 and a number 2 for client-2) for the resource to the arbiter via an individual channel (a and b).

The arbiter chooses non-deterministically between clients with pending requests, and then sends the number of the selected client-(1 or 2) via another channel (c) to the environment. A χ specification is given as follows:

$\langle$ disc x
, chan a, b, c
, x = 0
| $*(\partial_A(\mathrm{CLI_1} \parallel \mathrm{CLI_2} \parallel \mathrm{ARB}))$
$\rangle$,

$$\mathrm{CLI_1} \approx \mathrm{a}\,!!\,1, \quad \mathrm{CLI_2} \approx \mathrm{b}\,!!\,2,$$

$$\mathrm{ARB} \approx (\mathrm{a}\,??\,\mathrm{x} \,[\!]\, \mathrm{b}\,??\,\mathrm{x});\ \mathrm{c}\,!!\,\mathrm{x}.$$

The process terms $\mathrm{CLI_1}$, $\mathrm{CLI_2}$ and ARB model the behaviour of the client-1, client-2 and arbiter respectively as described above.

The encapsulation operator is applied to $\mathrm{CLI_1} \parallel \mathrm{CLI_2} \parallel \mathrm{ARB}$ to block some undesired internal send and receive actions (specified in the set A) via channels a and b. This means that only successful communication actions via channels a and b are allowed.

4.3 A simple arbiter

In general, the role of an arbiter is to grant access to the shared resource by raising the corresponding *grant* signal and keeping it that way until the *request* signal is removed.

A test for such an arbiter can be generated by an assertion as follows:

$$\text{“}assertion : grant \wedge request\text{”}.$$

If the assertion holds, this means that the arbiter work as expected.

Below is a χ specification of the simple arbiter:

$\langle$ disc $clk, clk_c, grant, request, t$
, $clk = clk_c = grant = request = \mathrm{false}, t = 0$
| INIT; $*(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER})$
$\rangle$, where

$$\begin{array}{rl}
\mathrm{INIT} & \approx clk, grant, request := \mathrm{false}, \mathrm{false}, \mathrm{false} \\
\mathrm{ARB} & \approx \mathrm{R_1};\ \mathrm{G};\ \mathrm{R_0} \\
\mathrm{R_1} & \approx \Delta 4;\ request := \mathrm{true} \\
\mathrm{G} & \approx \Delta 4;\ grant := \mathrm{true} \\
\mathrm{R_0} & \approx \Delta 4;\ request := \mathrm{false} \\
\mathrm{CLK} & \approx \Delta 5;\ clk_c, clk := clk, \neg clk \\
\mathrm{ASSER} & \approx \neg clk_c^{-} \wedge clk^{-} \wedge grant^{-} \wedge \\
 & request^{-} \rightarrow t := \mathtt{time}
\end{array}$$

The χ specification of the arbiter is a sequential composition of the process terms INIT and the repetition of the parallel composition of process terms ARB, CLK and ASSER:

- INIT - It assigns the initial values to variables clk, clk_c, $grant$ and $request$ (i.e. the initialisation).
- ARB - It models the change of behaviour of variables clk, clk_c, $grant$ and $request$ according to time.
- CLK - It models the behaviour of a clock (i.e. clk) which swaps the values between "false" and "true" every 5 time units.

- ASSER - It expresses the assertion for the arbiter (as indicated above). Also, it models the fact that the test of the assertion is executed whenever there is a positive change in clk. When this happens, the current time is assigned to the variable t.

In the process term CLK, variable clk_c is introduced (as a copy of clk_c) to save the temporary value of clk, which is used to model event change on the variable clk (i.e. event controls).

4.3.1 Hardware description of the arbiter

For those who are familiar with hardware description languages, the formal χ specification of the arbiter presented in Subsection 4.3 can be regarded as the mathematical model of the below hardware description of the arbiter in Verilog.

```
module assert();
reg clk, grant, request;
time current_time;
initial begin
   clk = 0;
   grant = 0;
   request = 0;
   #4 request = 1;
   #4 grant = 1;
   #4 request = 0;
   #4 $finish;
end
always #5 clk = ~ clk;
always @ (posedge clk)
begin
if (grant == 1 && request == 1)
    begin
    current_time = $time;
    $display
    {``working as expected'');
    end
end
endmodule
```

5 Analysis of χ Specifications

5.1 Hardware systems

This subsection briefly presents several analysis results of the hardware systems modelled in χ as shown in Section 4.

5.1.1 D flip-flop

The dynamic behaviour of the D flip-flop in χ was simulated using the χ simulator [26]. The simulation results are correlated to the behaviour of the D flip-flop as described in Section 4.1.

5.1.2 Asynchronous arbiter

The specification of the asynchronous arbiter was translated to the corresponding specification in mCRL [25] and then further verified the safety property: the mutual exclusion using CADP [22].

5.1.3 Simple arbiter

Using the deduction rules and properties of χ, the assertion property of the simple arbiter was proved by means of a complete mathematical proof.

5.2 Summary and other analysis techniques for χ

In this subsection, we survey various approaches that can be effectively used for the analysis of hardware systems described in χ (for different analysis purposes).

- Various properties (e.g. safety and liveness) can be proved by means of mathematical proofs using χ deduction rules and properties. However, this approach may not be intuitive to those who have not a strong Computer Science background, because rewriting of the specifications (based on the properties) and formal reasoning (based on the deduction rules) have to be made.

- In process algebras, *linearisation* is a transformation of a recursive specification into a linear representation, i.e., a kind of normal form that is convenient for many forms of analysis. Note that these linear representations are expressed as recursive specifications as well, but they use only a small subset of the full process algebra. In general, such linear representations can also be considered very compact representations of a possibly infinite state space. The original recursive specification and its transformation are required to be bisimilar, which ensures that the relevant specification properties are preserved. Some algorithms for linearisation of Hybrid Chi have already been developed (see [16] for details). These algorithms can be reasonable easily adopted for χ.

- χ can serve as a single-formalism-multi-solution, this means that we can translate a χ specification to the input languages of several verification tools (e.g. CADP, SPIN [23] and Uppaal [24])

and it can be verified in those verification tool environments. This work was reported in [17].

- We can use χ tools for simulation and verification of χ specifications. Below is a summary for χ tools:
  - χ *simulator:* a simulator for χ specifications was built (based on the formal semantics of χ);
  - χ *translators:* automatic translation tools, which convert χ specifications to the corresponding models/specifications in mCRL (the input language of CADP), Promela (the input language of SPIN) and timed automata (the input language of Uppaal);
  - *availability:* simulator, translation tools and manuals of χ can be found in [26].

6 Conclusions and Future Work

χ can be reasonably and effectively used to give formal specifications of hardware systems and possibly to analyse them using various analysis approaches as indicated in this paper.

In order to illustrate our work clearly, only simple hardware systems in χ were given in this paper. Nevertheless, the use of χ is generally applicable to all sizes and levels of hardware systems.

In our opinion, with respect to common hardware description languages (e.g. VHDL and Verilog), χ can precisely describe the behaviour of hardware systems in a complete mathematical way. Prior to χ, from literature, several process algebra based formalisms (e.g. CHP [15]) were used to give formal specifications of hardware systems.

In addition to those formalisms, χ enhances strength for formal specification, because χ has a comprehensive set of operators that enables process re-use, encapsulation, hierarchical and/or modular composition of processes, etc. Furthermore, χ has a rich set of support tools.

Recently, several other timed process algebras have been developed (e.g. $SystemC^{\mathbb{FL}}$ [28, 27, 29] and PAFSV [30]) that can also be used for formal specification and analysis of hardware systems.

Our future work will focus on:

- a comparative study between χ and other timed process algebras for the use of specification and analysis of hardware systems;
- the investigation of the applicability of existing techniques (e.g. genetic algorithms) to optimise χ specifications;
- the development of a χ synthesis-able subset based on the traditional technology mapping.

Availability

For research purposes, we would be pleased to receive interesting case studies on formal specification and analysis of hardware systems from anyone working in this area.

For more information, please send mail to pafesd@gmail.com or visit PAFESD web-site *http://digilander.libero.it/pafesd/*.

Acknowledgements

K.L. Man wishes to thank Jos Baeten, Bert van Beek, Mohammad Mousavi, Koos Rooda, Ramon Schiffelers, Pieter Cuijpers, Michel Reniers, Kees Middelburg, Uzma Khadim and Muck van Weerdenburg for many stimulating and helpful discussions (focusing on process algebras for distinct systems) in the past few years.

He is indebted to Andrea Fedeli for his significant contribution to the draft version of this paper.

Also, he would like to thank Michel Schellekens and Menouer Boubekeur for many stimulating and helpful discussions.

References:

[1] Baeten, J.C.M., Weijland, W.P., *Process algebra*, volume 18 of Cambridge tracts in Theoretical Computer Science, Cambridge University Press, Cambridge, United Kingdom, 1990.

[2] Baeten, J.C.M., Middelburg, C.A., *Process algebra with timing*, EATCS Monographs Series, Springer-Verlag, 2002.

[3] Nicollin, X., Sifakis, J., *The algebra of timed processes ATP: theory and application*, Information and Computation, 114(1):131-178, October, 1994.

[4] Cuijpers, P.J.L., Reniers, M.A., *Hybrid process algebra*, Journal of Logic and Algebraic Programming, 62(2):191-245, 2005.

[5] van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H., *Syntax and semantics of timed Chi*, Technical Report CS-Report 05-09, Eindhoven University of Technology, Department of Computer Science, The Netherlands, 2005.

[6] Man, K.L., Schiffelers, R.R.H., *Formal specification and analysis of hybrid systems*, PhD Thesis, Eindhoven University of Technology, The Netherlands, 2006.

[7] van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H., *Syntax and consistent equation semantics of hybrid Chi*, Journal of logic and algebraic programming, 68(1-2):129-210, 2006.

[8] Plotkin, G.D., *A structural approach to operational semantics*, Technical Report DIAMI FN-19, Aarhus University, Department of Computer Science, Denmark, 1981.

[9] *IEEE standard for Verilog hardware description language*, IEEE Std 1364-2005 (revision of IEEE Std 1364-2001), IEEE Computer Society, 2005.

[10] *IEEE standard for VHDL language reference manual*, IEEE Std 1076-2000 (incorporates IEEE Std 1076-1993 and IEEE Std 1076a-2000), IEEE Computer Society, 2000.

[11] Schneider, G., Qiwen, X., *Towards an operational semantics of Verilog*, UNU/IIST Report No. 147, International Institute for Software Technology, United Nations University, Macau, 1998.

[12] Breuer, P.T., Delgado Kloos, C., *Formal semantics for VHDL*, Kluwer Academic Publishers, 1995.

[13] Sasaki, H., Mizushima, K., Sasaki, T., *A formal semantics for Verilog-VHDL simulation interoperability by abstract state machine*, Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany, 1999.

[14] Huibiao, Z., Jifeng, H., *A DC-based semantics for Verilog*, UNU/IIST Report No. 183, International Institute for Software Technology, United Nations University, Macau, 2000.

[15] Salaun, G., Serwe, W., *Translating hardware process algebras into standard process algebras - illustration with CHP and LOTOS*, Proceedings of the International Conference on Integrated Formal Methods, Eindhoven, The Netherlands, 2005.

[16] Baeten, J.C.M., van Beek, D.A., Rooda, J.E., *Handbook of dynamic system modeling (chapter process algebra)*, CRC Press LLC, 2006.

[17] Bortnik, E.M., Trcka, N., Wijs, A.J., Luttik, B., van de Mortel-Fronczak, J.M., Baeten, J.C.M., Fokkink, W.J., Rooda, J.E., *Analyzing a Chi model of a turntable system using Spin, CADP and Uppaal*, Journal of logic and algebraic programming, 65(2):51-104, 2005.

[18] Davies, J., Schneider, S., *A brief history of Timed CSP*, Theoretical Computer Science, 138:183-235, 1995.

[19] Westerlund, T., Plosila, J., *Formal timing model for hardware components*, Technical Report 640 (tWePl04a), Turku Centre for Computer Science, Finland, 2004.

[20] Baeten, J.C.M., *A brief history of process algebra*, Technical Report CS-Report 04-02, Eindhoven University of Technology, Department of Computer Science, The Netherlands, 2004.

[21] Hoare, C.A.R., *Communicating sequential processes*, Communications of the ACM, 21(8):666-467, 1978.

[22] Fernandez, J.C., Garavel, H., Kerbrat, A., Mounier, L., Mateescu, R., Sighireanu, M., *CADP - a protocol validation and verification toolbox*, Proceedings of the 8^{th} Conference on Computer Aided verification, volume 1102 of Lecture Notes in Computer Science, pages 437-440, 1996.

[23] Holzmann, G.J., *The SPIN model checker*, Addison-Wesley, 2003.

[24] Larsen, K.G., Pettersson, P., Yi, W., *UPPAAL in a nutshell*, Journal on Software Tools for Technology Transfer, 1(1-2):134-152, 1997.

[25] mCRL homepage, *http://homepages.cwi.nl/~mcrl/*.

[26] Chi tools and manuals are available on *http://se.wtb.tue.nl/sewiki/chi/installation*.

[27] Man, K.L., $SystemC^{\mathbb{FL}}$*: Formalization of SystemC*, IEEE Proceedings of the Mediterranean Electrotechnical Conference, Dubrovnik, Croatia, 2004.

[28] Man, K.L., *Formal verification of* $SystemC^{\mathbb{FL}}$ *specifications using SPIN*, Proceedings of the 5th WSEAS International Conference on Microelectronics, Nanoelectronics and Optoelectronics, Prague, Czech Republic, 2006.

[29] Man, K.L., $SystemC^{\mathbb{FL}}$*: Formal specification and analysis of hardware/software co-designs*, Journal of The World Scientific and Engineering Academy and Society Transactions on Circuits and Systems, 3(5):361-368, 2006.

[30] Man, K.L., Boubekeur, M., Schellekens, M.P., *Process algebraic approach to SystemVerilog*, Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, Vancouver, British Columbia, Canada, April, 2007.